ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DISTRIBUIDO PARA CLASIFICACIÓN DE FICHAS LEGO BASADO EN IMÁGENES

SUBSISTEMA DE CONSULTA

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

EDWIN RAMIRO CABRERA CUICHÁN

edwin.cabrera@epn.edu.ec

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE

david.mejia@epn.edu.ec

DMQ, abril 2023

CERTIFICACIONES

Yo, EDWIN RAMIRO CABRERA CUICHÁN declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

EDWIN RAMIRO CABRERA CUICHÁN

Certifico que el presente trabajo de integración curricular fue desarrollado por EDWIN RAMIRO CABRERA CUICHÁN, bajo mi supervisión.

RAÚL DAVID MEJÍA NAVARRETE
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

EDWIN RAMIRO CABRERA CUICHÁN

RAÚL DAVID MEJÍA NAVARRETE

DEDICATORIA

A mi madre Mónica, a mi padre Misael y a mi hermana Magaly, por siempre estar pendientes y ser el motivo para sobresalir ante la adversidad.

AGRADECIMIENTO

No creo en la religión, pero sí creo en la bendición de mi madre por lo cual siempre estaré muy agradecido por estar de manera incondicional para mí. ¡Gracias mami!

Agradezco a mi padre por haberme formado con las costumbres y valores con las cuales hoy me desempeño en mi diario vivir y, sobre todo, por darme la pauta de poder darme cuenta de lo que está mal y tener la oportunidad de corregir.

Para describir la felicidad existe un sinnúmero de maneras, y mi hermana es la descripción para mí de lo que significa la felicidad. Gracias por existir. Eres mejor que yo.

Durante la carrera universitaria he compartido con muchas personas en las instalaciones de la EPN y en ese camino tuve el honor de compartir con personas que hicieron que mi estancia académica sea cálida. Luis Auqui, mi primer amigo de la Facultad. Andrés Santamaría, un gran ser humano de mi carrera. Aquí hago énfasis en mi agradecimiento a dos personas con quienes pude compartir muchas risas, trabajo, enojo, aprendizaje y nuevamente risas, gracias Naty e Iveth por haber compartido tiempo conmigo.

A los profesores de esta institución por sus enseñanzas, no solo académicas, sino también como ser humanos.

A mi tutor, David Mejía, por su paciencia y apoyo para dirigir este trabajo.

Finalmente, a la persona que ha venido a completar una de las etapas de mi vida, quien caminará junto a mi para alcanzar las metas, logros y alegrías propuestas en una historia sin fin, gracias Pamela.

ÍNDICE DE CONTENIDO

RESUMEN

En el presente Trabajo de Integración Curricular se presenta el diseño e implementación de un subsistema de consulta para un sistema de clasificación de fichas Lego basado en imágenes.

El subsistema de consulta permite presentar la información que ha sido almacenada en un subsistema de almacenamiento, información proveniente desde un subsistema de adquisición de imágenes. También permite consultar información de las fichas Lego que han sido clasificadas por el subsistema de clasificación.

El presente documento consta de 3 capítulos:

En el primer capítulo se resumen los conceptos referentes a las herramientas a usar como la interfaz de desarrollo y el motor de base de datos. También se menciona las tecnologías usadas como Entity Framework, para la interacción con la base de datos, usando el enfoque *Database First*; así como conceptos claves relacionados con ASP.NET MVC, Razor y Bootstrap Finalmente se concluye con la metodología ágil Kanban aplicada al desarrollo del subsistema de consulta.

En el segundo capítulo, se presenta un resumen del proceso de desarrollo del subsistema de consulta, con base en los requerimientos y las historias de usuario planteadas como parte del diseño. Adicionalmente, este capítulo contiene los *sketches* de las interfaces a implementar. También se hace mención del desarrollo de los *stubs* del subsistema de clasificación y del subsistema de almacenamiento, necesarios para probar el funcionamiento del subsistema de consulta.

El tercer capítulo contiene un resumen de las pruebas de funcionamiento y sus resultados obtenidos, así también se indican las correcciones realizadas con base en los resultados obtenidos en las pruebas. Finalmente se presentan las conclusiones y recomendaciones obtenidas al terminar el presente Trabajo de Integración Curricular.

Finalmente, en los Anexos se presenta los *sketches* de los diseños de las interfaces de usuario, el script de la base de datos, el código fuente correspondiente al subsistema de consulta, la encuesta realizada y sus resultados.

PALABRAS CLAVE: ASP.NET MVC, Razor, *Stub*, Entity Framework, Kanban.

ABSTRACT

This Curricular Integration Project presents the design and implementation of a query subsystem for a Lego chip classification system based on images.

The query subsystem allows presenting the information that has been stored in a storage subsystem, information coming from an image acquisition subsystem. It also allows querying information about Lego chips that have been classified by the classification subsystem.

This document consists of 3 chapters.

The first chapter summarizes the concepts related to the tools to be used such as the development interface and the database engine. It also mentions the technologies used such as Entity Framework, for interaction with the database, using the Database First approach; as well as key concepts related to ASP.NET MVC, Razor, and Bootstrap. Finally, it concludes with the Kanban methodology applied to the development of the query subsystem.

In the second chapter, a summary of the query subsystem development process is presented, based on the requirements and user stories raised as part of the design. Additionally, this chapter contains sketches of the interfaces to be implemented. It also mentions the development of stubs for the classification and storage subsystems, necessary to test the proper functioning of the query subsystem.

The third chapter contains a summary of the performance tests and their results obtained, as well as the corrections made based on the results obtained in the tests. Finally, the conclusions and recommendations obtained at the end of this Curricular Integration Project are presented.

Finally, the Annexes present the sketches of the user interface designs, the database script, the source code corresponding to the query subsystem, the conducted survey, and the results.

KEYWORDS: ASP.NET MVC, Razor, Stub, Entity Framework, Kanban.

1 INTRODUCCIÓN

En el presente Trabajo de Integración Curricular se describe el proceso realizado para desarrollar un subsistema de consulta, el cual es parte de un sistema distribuido para clasificación de fichas Lego basado en imágenes.

El subsistema se encargará de presentar la información de las fichas Lego almacenadas en la base de datos, así como la información de las imágenes clasificadas proporcionada por el subsistema de clasificación.

El subsistema, por medio de consultas, será capaz de presentar la información de las imágenes acorde a las características que son: color de la ficha, tipo de ficha y el factor. También será capaz de presentar la información de las imágenes clasificadas acorde a los parámetros que serán obtenidas en el subsistema de clasificación.

Como parte del subsistema se generarán *stubs,* tanto para el subsistema de almacenamiento como para el subsistema de clasificación que se encargarán de simular el funcionamiento de estos subsistemas y permitirán demostrar que el subsistema de consulta es funcional.

Para el desarrollo de este subsistema se usará el *framework* de Microsoft ASP.NET así como el patrón de desarrollo MVC (*Model View Controller*) definiendo tanto la lógica en los controladores, las validaciones en los modelos y la forma de presentar la información en las vistas. Para la interacción con la base de datos se implementará Entity Framework y se utilizará el método *Database First,* esto debido a que primero se diseñará la base de datos y posteriormente se inicia el desarrollo del código para el subsistema de consulta.

Para los estilos y el diseño responsivo de la interfaz de usuario se realizarán *sketches* que permitan esquematizar los mismos para luego ser implementados usando el *framework* Bootstrap.

Finalmente, se realizarán pruebas con los respectivos *stubs* del subsistema de clasificación y del subsistema de almacenamiento para verificar el correcto funcionamiento del subsistema y posteriormente analizar los resultados obtenidos.

1.1 Objetivo general

Desarrollar el subsistema de consulta como parte del sistema distribuido para clasificación de fichas Lego basado en imágenes.

1.2 Objetivos específicos

Para el presente Trabajo de Integración Curricular los objetivos específicos son:

1. Analizar las tecnologías y herramientas necesarias para el desarrollo del subsistema de consulta.
2. Diseñar la lógica de los controladores, modelos y vistas para el correspondiente intercambio de información con el subsistema de clasificación y el subsistema de almacenamiento.
3. Implementar el subsistema haciendo uso del *framework* ASP.NET MVC, Entity Framework y Bootstrap, y con base en el diseño establecido.

1.3 Alcance

El presente Trabajo de Integración Curricular consiste en el desarrollo de un subsistema de consulta como parte del proyecto: “Desarrollo de un sistema distribuido para clasificación de fichas Lego basado en imágenes”. El subsistema permitirá presentar la información que, el subsistema de clasificación ha generado, una vez que el usuario ha ingresado una imagen por medio del subsistema de adquisición. Así también, permitirá consultar la información almacenada por el subsistema de almacenamiento y presentarla en la interfaz de usuario.

Para la búsqueda, se usará como criterios el color y la forma de la ficha de Lego. Para comprobar el funcionamiento e independencia del subsistema de clasificación, se implementarán 2 *stubs*, los cuales emularán al subsistema de clasificación y al subsistema de almacenamiento, respectivamente.

El subsistema, de manera general, dispondrá de dos vistas principales.

- La vista que presenta la información de las imágenes almacenadas. Aquí se presentará la imagen que inicialmente fue cargada al subsistema de adquisición, junto con la información de la imagen cargada.
- La vista de consulta se encargará de presentar la información que se encuentre almacenada en el subsistema de almacenamiento y está relacionada con el subsistema de clasificación. Estás consultas se las puede realizar usando como criterios el color, tipo de la ficha Lego y factor.

Posteriormente se solicitará a 10 usuarios que prueben el subsistema de clasificación, luego de lo cual se aplicará una encuesta para determinar su opinión respecto a su funcionamiento.

Se corregirá los errores que se determinen como resultado de las pruebas del subsistema de consulta.

1.4 Marco Teórico

En esta sección se describe la teoría, tecnologías y herramientas utilizadas en este Trabajo de Integración Curricular.

1.4.1 ASP.NET MVC

ASP.NET MVC es un *framework* desarrollado por Microsoft que combina la potencia del patrón de desarrollo MVC con las técnicas e ideas del desarrollo ágil y aprovecha las características de ASP.NET.

Se considera un *framework* como una estructura o base que se puede usar para desarrollar una aplicación o sitio web de una manera más fácil y eficiente [1].

MVC es un patrón de diseño de software que permiten separar los componentes de una aplicación en tres capas distintas.

Las aplicaciones que emplean el patron MVC contienen [2]:

- **Modelos:** Se definen como clases que representan los datos que se utilizan en una aplicación. Contienen lógica de negocio.
- **Vistas:** Son archivos de plantilla encargados de presentar la información, generando dinámicamente respuestas HTML[1] (*Hypertext Markup Language*) y la interfaz que se mostrará al usuario final.
- **Controladores:** Son clases que se encargan de recibir las solicitudes provenientes desde el cliente, procesarlas y coordinar la interacción entre la vista y el modelo.

[1] HTML: Es un lenguaje de marcado utilizado para crear páginas web y otros documentos que se pueden visualizar en un navegador web.

Los modelos se relacionan directamente con los datos y estos representan objetos que se usan en la aplicación web. Estas clases tienen la posibilidad de incluir constructores y métodos que se encarguen de la lógica de negocio de ser necesario. Las aplicaciones web poseen persistencia de datos y en las aplicaciones desarrolladas con el patrón MVC el modelo es el encargado de incluir código para leer y escribir registros en la base de datos.

En ASP.NET MVC, para acceder a los datos se puede usar varios *frameworks,* pero el más común para realizar esta tarea es Entity Framework.

Un controlador en ASP.NET es una clase .NET que se encarga de procesar y responder a las solicitudes del usuario en una aplicación web MVC. Normalmente, por cada clase modelo existe una clase controlador.

Los controladores son derivados de la clase base `System.Web.Mvc.Controller` y contienen métodos que se ejecutan para producir una respuesta a una solicitud del usuario. Estos métodos, también conocidos como métodos de acción o acciones, pueden retornar un resultado del tipo `ActionResult` [3].

La vista es el componente encargado de representar la información que se debe presentar al usuario en respuesta a una solicitud.

En las aplicaciones desarrolladas con ASP.NET MVC, las peticiones entrantes son manejadas por los controladores y estas solicitudes son asignadas a métodos de acción de estos controladores.

Un método de acción de un controlador, por ejemplo, puede devolver una vista o devolver cualquier tipo de acción como dirigir esa solicitud, a otro método de acción del mismo u otro controlador.

Por esta razón, las vistas están pensadas para encapsular la lógica de presentación. Es decir, no deben de contener la lógica de la aplicación ni el código de conexión con bases de datos.

La vista es un archivo de extensión HTML con código C# que se utiliza para definir la estructura y la apariencia visual de la página web. A menudo, la vista utiliza una sintaxis especial llamada Razor que permite combinar código C# con HTML. En ASP.NET MVC, la extensión de archivo que se utiliza para las vistas es `.cshtml`.

Cuando llega el momento de devolver la respuesta, ASP.NET MVC buscará la vista con el nombre establecido en el controlador.

Normalmente, para generar las vistas, se emplea el método auxiliar `View` el cual obtendrá el `ViewResult` correspondiente. Cuando se invoca al método `View`, el desarrollador indica a ASP.NET MVC que busque una vista con el mismo nombre que la acción del controlador actual. El fragmento de código de la Figura 1.1, muestra el código un método de acción denominado `Index`. Este método, que no recibe parámetros, presentará un mensaje definido en la propiedad `ViewBag.Message`, como se indica en la línea 3. La línea 4, emplea el método `View` para devolver una vista con el mismo nombre del método de acción.

```
1 public ActionResult Index()
2 {
3   ViewBag.Message = "Your app description page.";
4   return View();
5
6 }
7
```

Figura 1.1. `ActionResult` que retorna una Vista

Para que en el lado del servidor se ejecute el código de programación y a través de este se genere el código HTML que el navegador web del cliente pueda presentar, debe existir un motor de vistas que se encargue de interpretar los archivos que contienen las vistas de una aplicación web MVC.

En ASP.NET MVC, el motor de vistas por defecto es Razor, el cual se encarga de identificar el código escrito en C# en el lado del servidor buscando entre las líneas de la vista aquellas que contengan el símbolo `@`.

ASP.NET MVC llama a distintas clases de controlador con sus respectivos métodos de acción acorde a la dirección URL[2] (*Uniform Resource Locator*) correspondiente al recurso solicitado por el usuario [2].

El proceso que sigue ASP.NET MVC para atender a una solicitud y generar una respuesta se presenta en la Figura 1.2. Acorde a la Figura 1.2, el navegador del cliente envía una

[2] ULR: Representa a una cadena de caracteres utilizada para identificar de manera exclusiva un recurso en internet.

solicitud HTTP[3] (*Hypertext Transfer Protocol*) al servidor web que aloja a la aplicación ASP.NET. El enrutador de ASP.NET MVC determina que controlador y método de acción deben manejar la solicitud en función de la URL y de las rutas registradas en la tabla de enrutamiento. El controlador recibe la solicitud HTTP y, en función de la acción solicitada, realiza las operaciones necesarias para obtener y manipular los datos de la aplicación. El controlador selecciona la vista adecuada y utiliza el método `View` para devolver la vista al cliente. La vista es creada usados los datos provenientes del controlador y se transmite al cliente como una respuesta HTTP. El cliente recibe la respuesta HTTP y presenta la vista en el navegador.

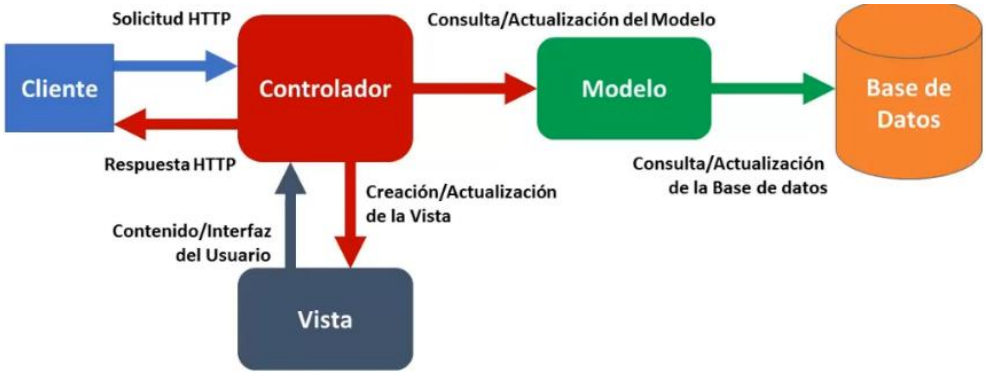

Figura 1.2. *Framework* ASP.NET MVC (Modelo – Vista – Controlador)

1.4.2 Entity Framework

Entity Framework es un ORM (*Object Relacional Mapper*) desarrollado por Microsoft, que permite trabajar con datos relacionales tratándolos como objetos sin la necesidad de ocuparse de las tablas y columnas de la base de datos. Además, proporciona un sistema basado en modelos que hace que la creación de una capa de acceso a datos sea sencilla.

Facilita la tarea de crear una capa de acceso a los datos y acceder a los mismos, al representar los datos como un modelo conceptual, es decir, un conjunto de entidades y relaciones. Dado que las tablas de la base de datos no pueden tener relaciones avanzadas (herencia) como las entidades del dominio, el modelo de negocio, es decir, el modelo conceptual se puede utilizar para adaptarse al dominio de la aplicación utilizando relaciones entre las entidades [4, p. 8].

[3] HTTP: Es un protocolo de comunicación utilizado para transferir datos a través de internet.

Entity Framework está escrito sobre el marco ADO.NET por lo que todavía usa los métodos y clases de ADO.NET para realizar las operaciones de datos. En la Figura 1.3 se presenta la arquitectura de Entity Framework.

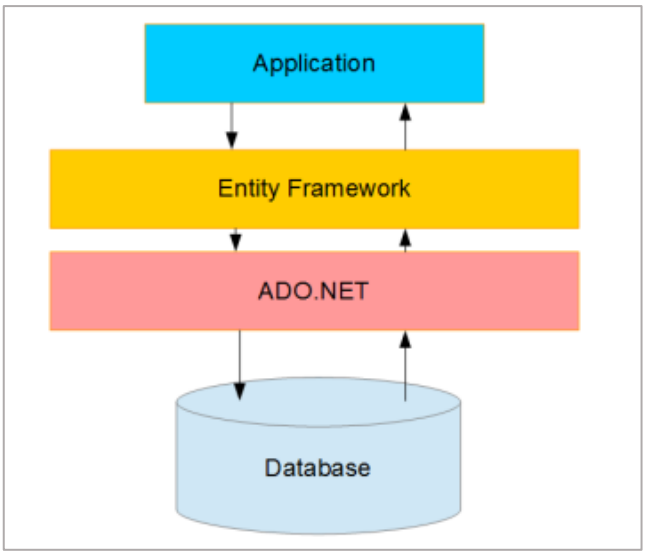

Figura 1.3. Arquitectura de Entity Framework

Al emplear Entity Framework, es necesario crear un modelo de datos conceptual conocido como EDM (*Entity Data Model*), que es el componente central del marco de trabajo.

El EDM incluye definiciones para las clases de los modelos conceptuales, las relaciones entre ellas y como se asignan los modelos al esquema de la base de datos.

Luego de haber creado que el EDM, se puede llevar acabo las operaciones CRUD (*Create, Read, Update & Delete*) y Entity Framework se encargará de convertir estas consultas de objetos en consultas de base de datos.

Después de ejecutar las consultas, Entity Framework convertirá los resultados en instancias de objeto de modelo conceptual. Para realizar esta traducción de consultas de objetos a consultas SQL y datos relacionales a modelos conceptuales, Entity Framework utilizará la información de asociación almacenada en el EDM [4, p. 9].

Uno de los enfoques de desarrollo de aplicaciones que utiliza el marco de trabajo de .NET Entity Framework para generar automáticamente el código de la aplicación a partir de una base de datos existente es *Database First*.

En este enfoque, se comienza por diseñar y crear la base de datos, y luego se utiliza Entity Framework para generar el modelo de objetos y el código de la aplicación [5].

1.4.3 Razor

Es una sintaxis y un motor de plantillas utilizado en ASP:NET MVC, que permite combinar código y contenido de manera fluida y expresiva. Razor permite al programador escribir código utilizando lenguajes como C# o Visual Basic .NET combinado con el lenguaje de marcado HTML.

Razor proporciona dos opciones para distinguir entre el lenguaje de programación y el lenguaje de marcado en un archivo de vista: *code nugget* y *code blocks*.

Los *code nugget* son expresiones simples que se evalúan y representan en línea. La expresión comienza inmediatamente después del símbolo `@`, como se presenta en la Figura 1.4. Los *code nugget* deben devolver lenguaje de marcado para que la vista se represente.

Figura 1.4. Salida de un *code nugget*

Un *code block,* es una sección de la vista que contiene código en lugar de una combinación de marcado y código. En la Figura 1.5, entre las líneas 1 y 4, se presenta como Razor define un *code block* como cualquier sección de una plantilla de Razor envuelta en caracteres `@{}`.

Figura 1.5. Ejemplo de *code block*

Los *code blocks* no presentan nada en la vista. En su lugar, le permiten escribir código arbitrario que no requiere valor de retorno. Las variables definidas dentro de los *code blocks* pueden ser utilizadas por los *code nugget* en el mismo ámbito [6].

En Razor, una plantilla compartida (también conocida como `layout`) es una plantilla HTML que se utiliza como base para múltiples vistas en una aplicación web. La plantilla define la estructura básica de la página, como la estructura HTML, las hojas de estilo CSS, los scripts JavaScript y cualquier otro contenido compartido en todas las vistas.

La plantilla compartida denominada `_Layout.cshtml`, se localiza en la carpeta `Views/Shared` de una aplicación ASP.NET MVC.

```
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5      <title>@View.Title</title>
6    </head>
7    <body>
8      <div class="header">
9        @RenderSection("Header")
10     </div>
11     @RenderBody()
12
13     <div class="footer">
14       @RenderSection("Footer")
15     </div>
16   </body>
17 </html>
```

Figura 1.6. Estructura básica del archivo `_Layout.cshtml`

En la Figura 1.6, se presenta la estructura de una vista compartida `_layout`. En las líneas 9, 11 y 14 se indica como el diseño se basa en variables y funciones auxiliares como `@RenderSection` y `@RenderBody` para definir secciones como cabecera, contenido principal y pie de página respectivamente, en una página y utilizarla en múltiples diseños de página.

Una vez definido el archivo base de diseño de Razor, las vistas hacen referencia al diseño base y proporcionan contenido para las secciones definidas dentro del nuevo diseño.

La Figura 1.7, en la línea 1, se presenta como se hace referencia dentro de una vista de contenido básico, al diseño base `_Layout.cshtml`.

```
@{ Layout = "~/_Layout.cshtml"; }

@section Header {
<h1>EBuy Online Action Side<h1>
}

@section Footer {
    Copyright @DateTime.Now.Year
  }
<div class="main">
    This is the main content.
</div>
```

Figura 1.7. Referencia al archivo `_Layout.cshtml`

1.4.4 Bootstrap

Es un *framework* de CSS (*Cascading Style Sheet*) para desarrollar sitios web adaptables (*responsive*) y de apariencia uniforme. Bootstrap incluye un conjunto de clases escritas mediante CSS que permiten personalizar el estilo de los elementos HTML en el sitio web que se desea implementar [1].

Figura 1.8. Sitio web responsive

Se considera que un sitio web es responsivo cuando se adapta a distintos dispositivos acorde a su tamaño y orientación de la pantalla, esto en el contexto del desarrollo web (ver Fig. 1.8). El desarrollo de sitios web con Bootstrap se fundamenta en tres pilares: grilla, componentes e íconos.

- Grilla: Permite ver cómo se van a estructurar los distintos elementos de la página web y cómo se van a adaptar acorde al tamaño del dispositivo.
- Componentes: Son similares a elementos HTML reutilizables que vienen con estilos predeterminados.

- Íconos: Es una biblioteca de íconos en formato `SVG`[4] (*Scalable Vector Graphics*) disponibles con una licencia de código abierto del MIT (Massachusetts Institute of Technology) [7].

Una de las ventajas de Bootstrap es que permite a los desarrolladores crear prototipos rápidos y efectivos, reduciendo el tiempo de desarrollo y, por lo tanto, los costos.

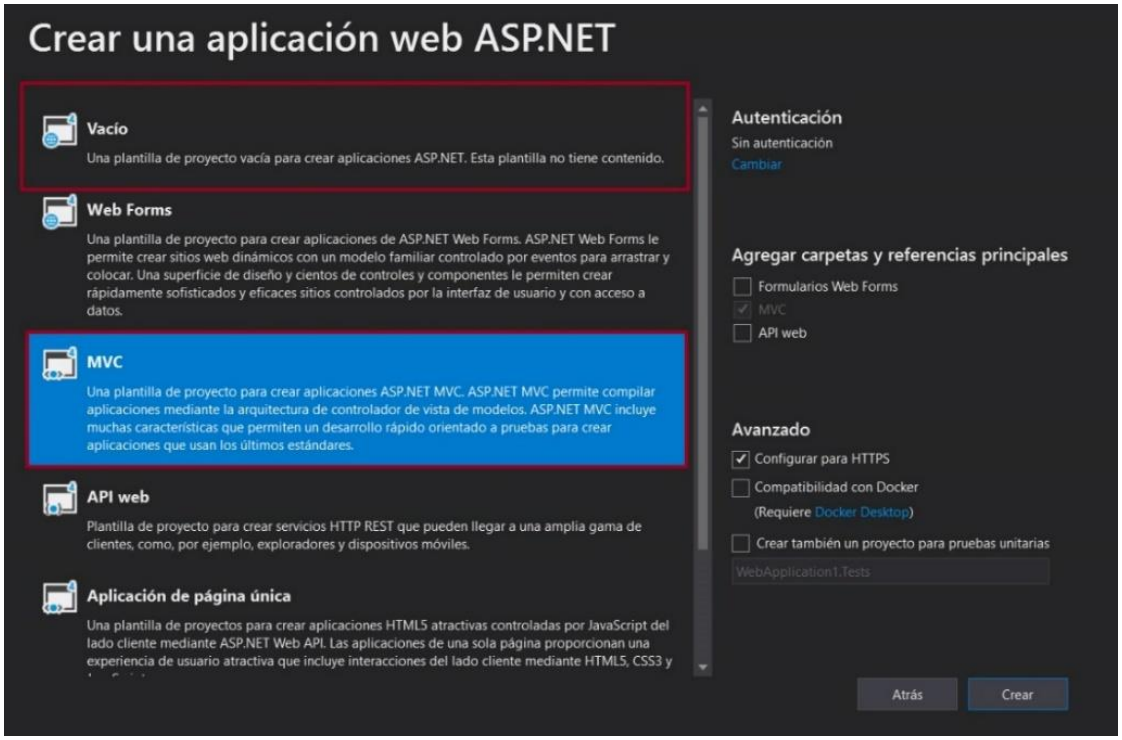

Figura 1.9. Plantillas para crear una aplicación web ASP.NET

Al crear una aplicación con ASP.NET en Visual Studio, la instalación automática de Bootstrap dependerá acorde al tipo de plantilla que se seleccione.
Si se selecciona la plantilla `Vacío` con las carpetas y referencias principales de MVC, Bootstrap no se instalará. Ocurre lo contrario si se selecciona la plantilla de MVC.

1.4.4 Metodología Kanban

Es un enfoque de gestión de procesos y proyectos que se basa en el uso de tableros Kanban (ver Figura 1.10) para visualizar el flujo de trabajo y limitar el trabajo en progreso.

[4] SVG: Formato de archivo utilizado para representar gráficos vectoriales en la web.

El objetivo de este método es identificar posibles cuellos de botella y corregirlos para que el desarrollo del trabajo sea óptimo.

Para hacer un buen uso de esta metodología y maximizar los beneficios de un proyecto o proceso, y así mejorar el flujo, se deben seguir los siguientes pasos [8]:

1. Presentar el flujo de trabajo. Consiste en el uso del tablero Kanban que representa visualmente el flujo de trabajo, las etapas y las tareas.
2. Limitar los trabajos en proceso. Estas son todas las tareas en las cuales todo el equipo está trabajando actualmente lo cual podría detener el trabajo de los demás grupos.
3. Administrar el flujo de trabajo. Se debe resaltar las distintas etapas y sobre todo las etapas actuales de los trabajos que se están realizando.
4. Hacer explícitas las reglas del proceso.
5. Implementar reuniones de retroalimentación.

Figura 1.10. Tablero Kanban

Product Backlog también conocido como Lista de Producto es un registro organizado de todos los elementos que podrían ser necesarios para el producto, y constituye la única fuente de requisitos para cualquier modificación que se realice en el mismo [9].

2 METODOLOGÍA

La metodología empleada en el presente Trabajo de Integración Curricular fue la metodología ágil Kanban. Aquí se han definido cada una de las tareas a desarrollar. Posteriormente, estas tareas fueron colocadas en el tablero Kanban especificando las siguientes etapas: por realizar, en proceso y finalizado.

Como parte del diseño, por medio de una entrevista realizada al director a cargo del presente Trabajo de Integración Curricular, se definieron las historias de usuario y los requerimientos del subsistema a ser desarrollado. Con la información obtenida, se identificaron las tareas del *product backlog*, que serán presentadas acorde a su importancia dentro del desarrollo del subsistema de consulta.

Con la información ordenada, se precedió a diseñar el subsistema de consulta, identificando la arquitectura de software a emplear, implementando la base de datos y definiendo los *skecthes* de las interfaces de usuario a desarrollar, así como, las características principales de las mismas.

Las herramientas empleadas en el desarrollo del subsistema fueron: el IDE (*Integrated Development Enviroment*) Microsoft Visual Studio Community 2019 versión 16.11.6, Microsoft SQL Server Management Studio 18.

El software Balsamiq Wireframes en su versión 4.6, fue empleado para diseñar los *sketches* de la interfaz de usuario.

Para el desarrollo del subsistema de consulta con el *framework* ASP.NET MVC, se empleó la versión .NET Framework 4.7.2. Para la interacción con la base de datos se empleó la versión de Entity Framework 6.1.3 y para la interfaz de usuario se usó la versión de Bootstrap 4.3.1.

Para la instalación o actualización de estos *framework* dentro del desarrollo del subsistema, se empleó el administrador de paquetes NuGet que está incluido en el IDE mencionado.

El enfoque empleado por Entity Framework para la interacción con la base de datos y los modelos es *Database First,* debido a que se inicia por el diseño y desarrollo de la base de datos.

Como parte de la implementación del subsistema de consulta, se desarrollaron *stubs* del subsistema de almacenamiento y el subsistema de clasificación para verificar el funcionamiento y realizar las pruebas pertinentes.

Las pruebas son: presentar la información de la base de datos referente a las características de las imágenes almacenadas y clasificadas, realizar consultas en función a las características de la ficha Lego como color, tipo y factor. Finalmente, que el diseño de la interfaz de usuario sea responsivo.

2.1 Historias de Usuario

Con base a la información proporcionada por el director del Trabajo de Integración Curricular a través de una entrevista, se obtiene los requerimientos funcionales y requerimientos no funcionales del subsistema de consulta.

2.1.1 Requerimientos funcionales

- El subsistema debe permitir la visualización de las imágenes almacenadas en la base de datos.
- El subsistema debe permitir la visualización de las imágenes clasificadas.
- El subsistema debe permitir realizar la búsqueda de imágenes clasificadas acorde al color, tipo o factor de una ficha Lego.

2.1.2 Requerimientos no funcionales

- El subsistema será realizado, usando el *framework* ASP.NET MVC 5.
- La base de datos con la que el subsistema de consulta debe interactuar será una base de datos relacional.
- Para la manipulación de la información en la base de datos el subsistema debe emplear Entity Framework con el enfoque Database First.
- El subsistema empleará el lenguaje de consulta LINQ para extraer la información de las imágenes en la base de datos.
- El subsistema debe presentar un comportamiento responsivo empleando el *framework* Bootstrap.
- El subsistema se debe ejecutar sobre un servidor IIS (*Internet Information Service*).

Las historias de usuario generadas se presentan en la Tabla 2.1 la cual tiene los siguientes campos: Identificador, Nombre y Descripción.

Un ejemplo del formato del identificador es `HU_01`.

Tabla 2.1. Historia de Usuarios

Identificador	Nombre	Descripción
HU_01	Presentar la información de las imágenes almacenadas	Es usuario podrá visualizar las imágenes almacenadas que han sido almacenadas en la base de datos.
HU_02	Presentar la información de las imágenes clasificadas	El usuario podrá visualizar la información de las imágenes clasificadas.
HU_03	Realizar consultas con base a los atributos de las fichas Lego	El usuario podrá hacer consultas en de las fichas Lego acorde al color, tipo y factor.
HU_04	La aplicación debe tener comportamiento responsivo	El usuario podrá visualizar la aplicación en dispositivos con distintos tamaños de pantalla.

2.1.3 Product backlog

Las actividades que forma parte del producto backlog fueron obtenidas con base a la información proporcionada por el director del Trabajo de Integración Curricular.

En la tabla 2.2 se detalla el *producto backlog*.

Tabla 2.2 *Product backlog*

No.	Tareas
1	Diseño y ejecución de la base de datos.
2	Diseño de los *sketches* de las interfaces de usuario.
3	Implementación de Entity Framework y codificación de los modelos.
4	Implementación de la funcionalidad para visualizar las imágenes almacenadas en la base de datos.
5	Implementación de la funcionalidad para visualizar las imágenes clasificadas.
6	Desarrollo de la funcionalidad de búsqueda acorde a parámetros de búsqueda como color, tipo y factor de la ficha Lego.

2.2 Diseño

La implementación del subsistema de consulta se la realizó usando el patrón MVC. la arquitectura Cliente – Servidor y una base de datos relacional.

Como parte del Trabajo de Integración Curricular se generó un *stub* para el almacenamiento de la información de las imágenes. Debido a este *stub* se diseña e implementa la base de datos y se emplea Entity Framework con el enfoque Database First para gestionar la información con el subsistema de consulta.

2.2.1 Diseño de la base de datos

En el diseño de la base de datos, se identificaron las principales características de las imágenes a tratar y de las fichas que se clasificarán. Las tablas obtenidas son:

- `tblLego`
- `tblImagen`
- `tblTipoLego`

La relación que existe entre las tablas y entre los atributos se puede observar en el diagrama relacional de la Figura 2.1.

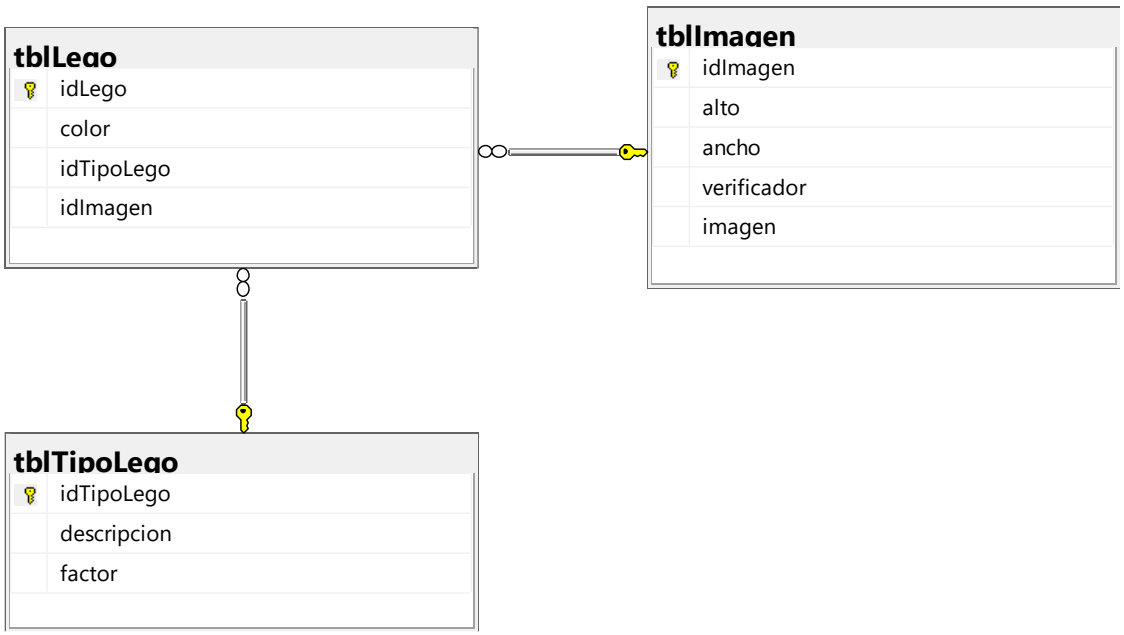

Figura 2.1. Diagrama relacional

2.2.2 Diseño de Clases

En la Figura 2.2. se presenta un diagrama de clases correspondiente al subsistema de consulta.

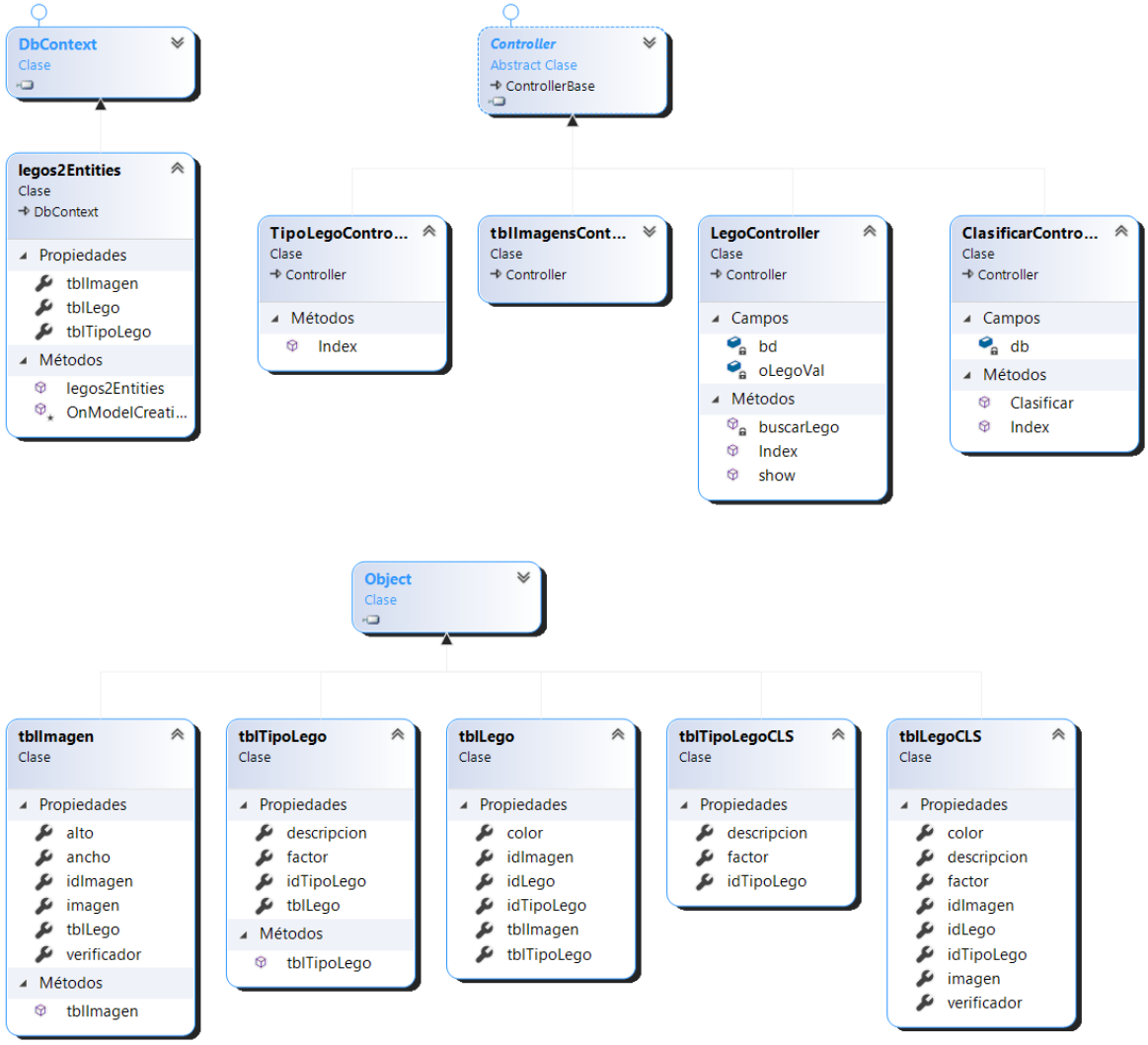

Figura 2.2. Diagrama de clases

La clase `legos2Entities` es generada por Entity Framework y es donde se define la conexión con la base de datos. Esta clase deriva de la clase base `DBContext`.

Las clases de los modelos generados por Entity Framework son `tblImagen`, `tblTipoLego` y `tblLego`. Estas clases se asocian con las tablas en la base de datos. Adicionalmente se implementan dos clases modelo denominadas `tblTipoLegoCLS` y `tblLegoCLS` que son usados para representar datos adaptándose a la información requerida por las vistas. Estas 5 clases se derivan de la clase base `Object`.

Los controladores implementados en el subsistema de consulta son: `TipoLegoController`, `tblImagensController`, `LegoController` y `ClasificarController`. Estas clases se derivan de la clase abstracta `Controller`. Las clases `tblImagensController` y `TipoLegoController` son las encargadas de generar extraer la información de las imágenes almacenadas y de los tipos de lego almacenadas en la base de datos. El controlador `LegoController` es el encargado de gestionar la información de las imágenes clasificadas y filtrar la información acorde a las características de las fichas Lego.

Finalmente, el controlador `ClasificarController` ha sido desarrollado como parte del *stub* del subsistema de clasificación.

El nombre de las clases que representan los controladores, por convención, llevan el sufijo `Controller`. En el presente Trabajo de Integración Curricular desde este punto, para referirse a los controladores, ya no se mencionará el sufijo `Controller`.

2.2.3 Sketches para las vistas

Como parte del diseño del subsistema de consulta, se realizaron los *sketches,* que han sido implementados con base a los requerimientos para subsistema. El objetivo es tener un esquema general referente a las interfaces de usuario producidas por las vistas. Se presentan 3 *sketches* que se describen a continuación.

Todos los *sketches* diseñados para el subsistema se encuentran en el ANEXO II.

En la Figura 2.3, se presenta el *sketch* denominado `INICIO`. Este *sketch* corresponde a la vista que presentará la página de inicio del subsistema de consulta. El *sketch* está conformado por un menú lateral en el lado izquierdo que posee las distintas opciones de navegación entre las interfaces de usuario dentro del subsistema de consulta que son `IMÁGENES ALMACENADAS`, `TIPOS DE LEGO`, `CLASIFICAR` e `IMÁGENES CLASIFICADAS`.

En la Figura 2.4 se presenta el *sketch* denominado `IMÁGENES ALMACENADAS`. Este *sketch* corresponde a la vista que permitirá presentar, a manera de lista, la información correspondiente a las imágenes que han sido almacenadas en la base de datos con sus respectivos atributos como alto y ancho. El campo `¿Es Lego?` de la lista, presentará el contenido del atributo `verificador` de la tabla `tblImagen`. Este atributo permite

identificar si la imagen se considera como ficha Lego o no, y es parte del *stub* del subsistema de clasificación.

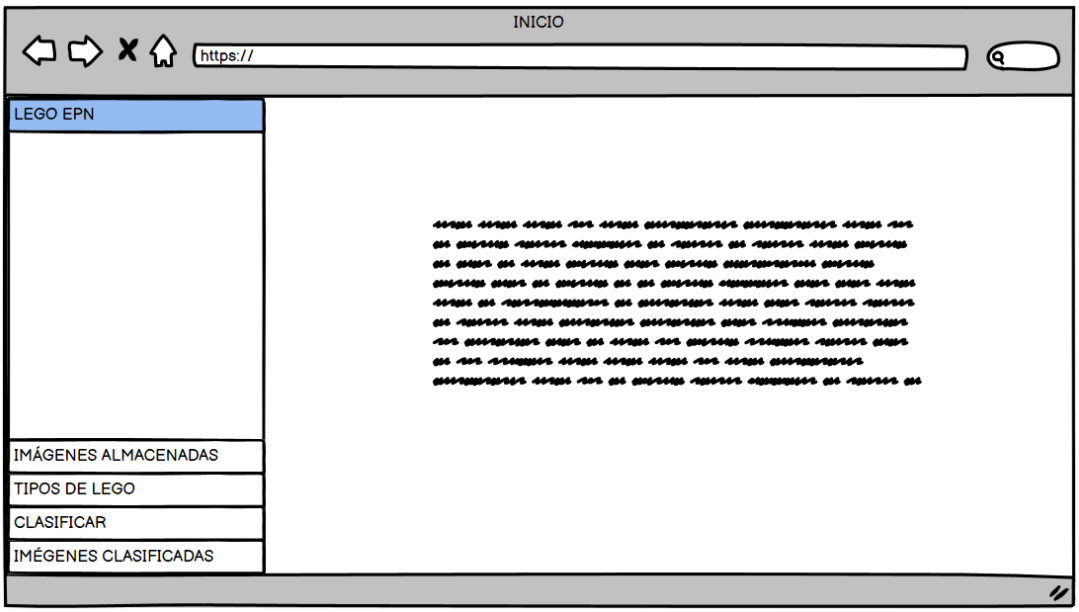

Figura 2.3. *Sketch* `INICIO`

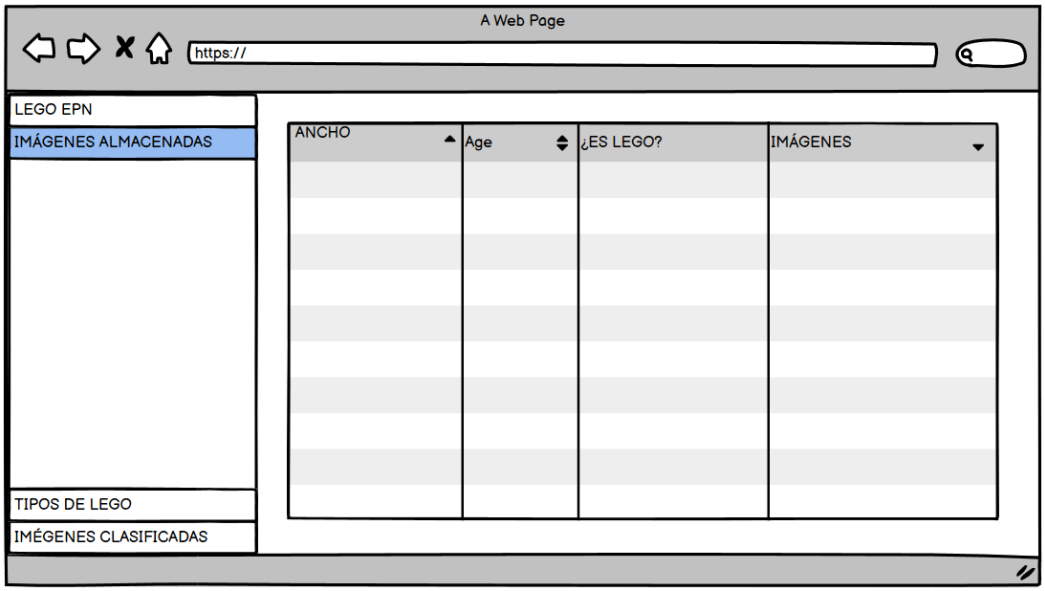

Figura 2.4. *Sketch* `IMÁGENES ALMACENADAS`

Finalmente, en la Figura 2.5 se presenta el *sketch* denominado `IMÁGENES CLASIFICADAS` correspondiente a la vista que se encarga de presentar en una lista las

imágenes que han sido clasificadas. Además, consta de cuadros de texto, que permiten realizar la búsqueda de información relacionada con las fichas Lego. Estos atributos de búsqueda son: color, tipo y factor.

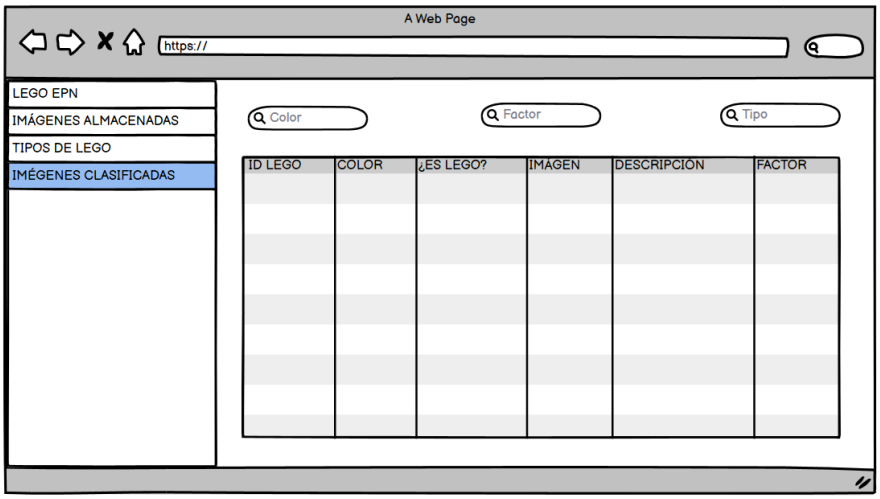

Figura 2.5. *Sketch* `IMÁGENES CLASIFICADAS`

2.3 Implementación

El subsistema de consulta del presente Trabajo de Integración Curricular inició desde el diseño e implementación de la base de datos la cual ha sido desarrollada por medio de un script que se puede verificar en el ANEXO I.

Previamente fue mencionado que, para trabajar con la información de la base de datos se utilizó Entity Framework quien nos generó clases modelo a partir de la base de datos existente implementando el enfoque *Database First*. Además, se implementaron otras clases modelo que llevan el sufijo `CLS` como convención para diferenciarlos de las clases modelo generadas por Entity Framework. Estás clases serán una combinación de los atributos de los modelos generados por Entity Framework cuyo objetivo es adaptarse a la representación de la información en la vista y añadir propiedades adicionales a los atributos de los modelos.

2.3.1 Modelos

En la Figura 2.6 se puede ver un fragmento del código del modelo `tblTipoLego` que ha sido generado por Entity Framework. Este modelo es el encargado de mapear los atributos de la tabla `tblTipoLego`. En las líneas de código 13, 14 y 15 el modelo define

las propiedades de un tipo de lego incluyendo su identificador, una descripción y un factor. El namespace `LegosProyecto.Models` de la línea 1, será el encargado de contener la definición del modelo para poder ser utilizada en el subsistema de consulta. En la 8 y 15, está presente una directiva encargada de suprimir determinadas advertencias emitidas por el compilador.

```
1  namespace LegosProyecto.Models
2  {
3      using System;
4      using System.Collections.Generic;
5
6      public partial class tblTipoLego
7      {
8          [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
   "CA2214:DoNotCallOverridableMethodsInConstructors")]
9          public tblTipoLego()
10         {
11             this.tblLego = new HashSet<tblLego>();
12         }
13         public int idTipoLego { get; set; }
14         public string descripcion { get; set; }
15         public string factor { get; set; }
16
17         [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
   "CA2227:CollectionPropertiesShouldBeReadOnly")]
18         public virtual ICollection<tblLego> tblLego { get; set; }
19     }
20 }
```

Figura 2.6. Fragmento del código del modelo `tblTipoLego`

La definición de los modelos generados por Entity Framework es similar cambiando los atributos por los que están conformados las tablas de la base de datos.

En la figura 2.7, se puede observar dentro de la carpeta `Models` del subsistema de consulta, las clases correspondientes a los modelos `tblLego`, `tblImagen` y `tblTipoLego` y las clases modelo que llevan el sufijo CLS.

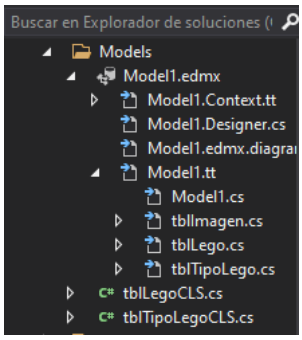

Figura 2.7. Contenido del directorio `Models`

En la Figura 2.8, se muestra la definición del modelo `tblLegoCLS`. Se implementa esta clase para definir un modelo de datos que se pueda usar en una vista para presentar información al usuario. La clase tiene propiedades de la tabla tblLego y de la tabla tblImagen. Estas propiedades están acompañadas del atributo `Display` de la línea 11 que se emplea para visualizar etiquetas legibles en la interfaz de usuario.

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel.DataAnnotations;
4  using System.Linq;
5  using System.Web;
6
7  namespace LegosProyecto.Models
8  {
9      public class tblLegoCLS
10     {
11         [Display(Name = "ID Lego")]
12         public int idLego { get; set; }
13         [Display(Name = "Color")]
14         public string color { get; set; }
15         public int idTipoLego { get; set; }
16         public int idImagen { get; set; }
17
18         [Display(Name ="¿Es Lego?")]
19         public Nullable<int> verificador { get; set; }
20         [Display(Name ="Imagen")]
21         public byte[] imagen { get; set; }
22
23         [Display(Name ="Descripción")]
24         public string descripcion { get; set; }
25         [Display(Name ="Factor")]
26         public string factor { get; set; }
27     }
28 }
```

Figura 2.8. Definición del modelo `tblLegoCLS`

2.3.2 Controladores

Los controladores principales dentro del subsistema de consulta son: `TipoLego`, `Lego` y `tblImagens`.

La Figura 2.9 corresponde el método `Index` del controlador `TipoLego`. Entre las líneas 4 y 13, el método es el encargado de extraer la información contenida en la tabla

`tblTipoLego`. Esta información es almacenada en una lista que posteriormente será enviada a la vista, que lleva el mismo nombre del método.

```
        public ActionResult Index()
        {
            List<tblTipoLegoCLS> listaTipoLego = null;
            using (var bd = new legos2Entities() )
            {
                listaTipoLego = (from tablaTipoLego in bd.tblTipoLego
                                                       select new tblTipoLegoCLS
                                                       {
                                                           idTipoLego = tablaTipoLego.idTipoLego,
                                                           descripcion = tablaTipoLego.descripcion,
                                                           factor = tablaTipoLego.factor
                                                       }).ToList();
            }
            return View(listaTipoLego);
        }
```

Figura 2.9. Método `Index` del controlador `TipoLego`

```
public ActionResult Index(tblLegoCLS otblLegoCLS)
{
    oLegoVal = otblLegoCLS;
    List<tblLegoCLS> listaLegos = null;
    //pongo otra variable
    List<tblLegoCLS> listaFiltrado;

    using (var bd = new legos2Entities())
    {
            listaLegos = (from tblLego in bd.tblLego
                          join tblImagen in bd.tblImagen
                          on tblLego.idImagen equals tblImagen.idImagen
                          join tblTipoLego in bd.tblTipoLego
                          on tblLego.idTipoLego equals tblTipoLego.idTipoLego
                          select new tblLegoCLS
                          {
                              idLego = tblLego.idLego,
                              color = tblLego.color,
                              verificador = tblImagen.verificador,
                              idImagen = tblImagen.idImagen,
                              descripcion = tblTipoLego.descripcion,
                              factor = tblTipoLego.factor
                          }).ToList();

        if(otblLegoCLS.color==null && otblLegoCLS.descripcion==null && otblLegoCLS.factor==null)
        {
            listaFiltrado = listaLegos;
        }
        else
        {
            Predicate<tblLegoCLS> pred = new Predicate<tblLegoCLS>(buscarLego);
            listaFiltrado = listaLegos.FindAll(pred);
        }
    }

    return View(listaFiltrado);
}
```

Figura 2.10. Método `Index` del controlador `Lego`

La Figura 2.10 corresponde al método de acción `Index` del controlador `Lego`. Entre las líneas 10 al 23, el método se encarga de extraer la información relacionada con las tablas `tblImagen`, `tblLego` y `tblTipoLego`. Esta información es almacenada en una lista de objetos de tipo `tblLegoCLS`. Este método recibe un parámetro que contiene los datos de entrada que serán filtrados con los datos de la lista.

Estos datos de entrada son inicializados en el método `buscarLego` de la Figura 2.11.

```
1  private bool buscarLego(tblLegoCLS otblLegoCLS)
2  {
3      bool busquedaColor = true;
4      bool busquedaDescripcion = true;
5      bool busquedaFactor = true;
6
7      if (oLegoVal.color != null)
8          busquedaColor =  otblLegoCLS.color.ToString().Contains(oLegoVal.color);
9      if (oLegoVal.descripcion != null)
10         busquedaDescripcion = otblLegoCLS.descripcion.ToString().Contains(oLegoVal.descripcion);
11     if (oLegoVal.factor != null)
12         busquedaDescripcion = otblLegoCLS.factor.ToString().Contains(oLegoVal.factor);
13
14     return (busquedaColor && busquedaDescripcion && busquedaFactor);
15 }
```

Figura 2.11. Método `buscarLego` del controlador `Lego`

El método `buscarLego` recibe como parámetro un objeto de tipo `tblLegoCLS` desde la vista. Este parámetro contiene información enviada desde la vista y será usada para inicializar los datos de entrada a ser usados en el método `Index`.

En la Figura 2.10, entre las líneas 25 a 33, con los datos de entrada inicializados se evalúa una condición. Si los datos de entrada están vacíos, el método Index devolverá a la vista, la lista completa de la información extraída de la base de datos. Si, por el contrario, los datos de entrada poseen algún valor correspondiente al color, tipo o factor; el método `Index` devolverá a la vista una lista filtrada acorde a las coincidencias de los datos de entrada.

Para presentar las imágenes extraídas de la base de datos, en el controlador `Lego` se implementa el método `show` de la Figura 2.12.

Entre las líneas de código del 1 al 5, el método `show` identifica a una acción del controlador `Lego` que será ejecutado en respuesta a una solicitud HTTP que contiene el parámetro `id`. El objetivo de esta acción es devolver la imagen correspondiente al `id` proporcionado como un archivo al navegador web. Este `id` servirá para filtrar la imagen en la tabla

`tblImagen` buscando una fila con la que coincida el `id` especificado. Una vez que la imagen se ha encontrado, se devuelve como un archivo utilizando el método `File`. En este método, como parámetro de debe incluir el formato de la imagen.

```
1 public ActionResult show(int id)
2 {
3     var imagenObtenida = bd.tblImagen.Where(x ⇒ x.idImagen == id).FirstOrDefault();
4     return File(imagenObtenida.imagen, "image/jpg");
5 }
```

Figura 2.12. Método `show` del controlador `Lego`

```
1  public ActionResult Clasificar(int id)
2  {
3      if (id == null)
4      {
5          return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
6      }
7      tblImagen oImagen = new tblImagen();
8      tblTipoLegoCLS otblTipoLegoCLS = new tblTipoLegoCLS();
9      tblLego otblLegoCLS = new tblLego();
10     var resultado = (from p in db.tblImagen
11                      where p.idImagen == id
12                      select p).SingleOrDefault();
13     resultado.verificador = 0;
14     db.SaveChanges();
15
16     otblLegoCLS.color = "rojo";
17     otblLegoCLS.idTipoLego = 4;
18     otblLegoCLS.idImagen = resultado.idImagen;
19
20     db.tblLego.Add(otblLegoCLS);
21     db.SaveChanges();
22     return RedirectToAction("Index","Lego");
23 }
```

Figura 2.13. Método `Clasificar` del controlador `Clasificar`

Finalmente, con respecto a los controladores, se hace referencia al controlador `Clasificar` correspondiente al *stub* del subsistema de clasificación. Este método

simulará la clasificación de una imagen, agregando valores aleatorios a los atributos relacionados con una imagen almacenada en la base de datos. La Figura 2.13 presenta el método `Clasificar` del controlador `Clasificar`. Este método es invocado cuando se envía una solicitud HTTP GET a la URL correspondiente. En este caso, la URL se espera que contenga un parámetro `id` que se utiliza para buscar una imagen en la base de datos. En las líneas del 3 al 6, se verifica si el parámetro `id` es nulo. Si es así, devuelve un error HTTP 400[5] *Bad Request*. En las líneas 10 al 12, se extrae la información de la tabla `tblImagen` para buscar la imagen con el `id` especificado. Una vez identificada la imagen, se establecen valores para los atributos presentes en las líneas 16 al 18. Se procede a almacenar esta información asignada en la base de datos y finalmente, se redirige al usuario a la acción `Index` del controlador `Lego`.

2.3.3 Vistas

Con los controladores implementados, se procede a codificar las vistas. La vista se crea a partir del método de acción, por lo cual lleva el mismo nombre.

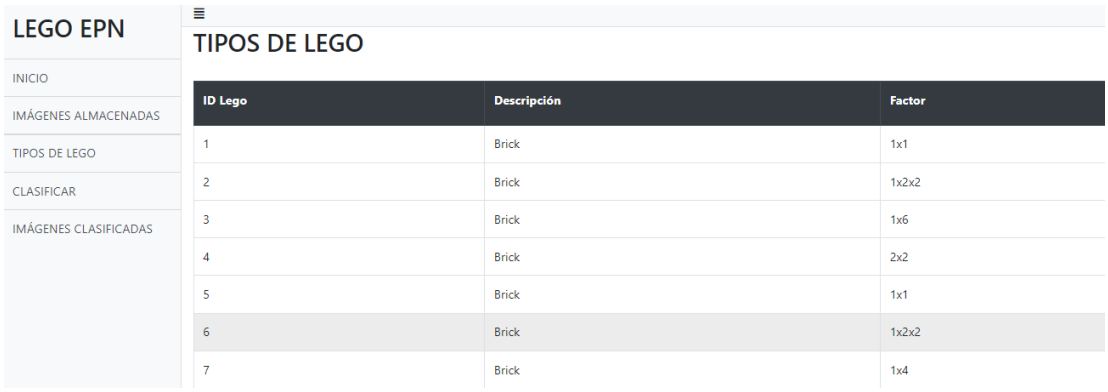

Figura 2.14. Interfaz de usuario de la vista *Index* asociada al controlador `TipoLego`

En la Figura 2.14 se presenta la interfaz de usuario generada por la vista `Index`, asociada al método de acción `Index`, perteneciente al controlador `TipoLego` donde se muestra un listado de la información que contiene la tabla `tblTipoLego` de la base de datos.

[5] HTTP 400: es un código de estado que indica que el servidor ha recibido una solicitud inválida del cliente y, como resultado, el servidor no puede o no procesará la solicitud.

La figura 2.15 presenta el código fuente de la vista `Index`. En la línea 1, se emplea la directiva `@using` para importar *namespace* `LegosProyecto.Models`, que contiene las definiciones de clase utilizada en el modelo. En la línea 2, `@model` se utiliza para definir el tipo de modelo `tblTipoLegoCLS` que se utilizará en la vista.

En las líneas de código del 3 al 6, se establece el título de la página con la propiedad `ViewBag.Title` y el diseño que se utilizará para la vista con la propiedad `Layout`. El diseño se especifica como una ruta de archivo a la carpeta `Views/Shared`, donde se pueden encontrar los archivos compartidos para la aplicación. La línea de código 12 genera una etiqueta HTML de encabezado de tabla que contiene el texto del nombre de los atributos contenidos en la lista.

```
1  @using LegosProyecto.Models
2  @model List<tblTipoLegoCLS>
3  @{
4      ViewBag.Title = "Index";
5      Layout = "~/Views/Shared/_Layout.cshtml";
6  }
7  <h2>TIPOS DE LEGO</h2>
8  <br />
9  <table class="table table-bordered table-hover">
10     <thead class="thead-dark">
11         <tr>
12             <th>@Html.LabelFor(p=>Model[0].idTipoLego)</th>
13             <th>@Html.LabelFor(p=>Model[0].descripcion)</th>
14             <th>@Html.LabelFor(p=>Model[0].factor)</th>
15         </tr>
16     </thead>
17     <tbody>
18         @foreach(var item in Model)      {
19                 <tr>
20                     <td>@item.idTipoLego</td>
21                     <td>@item.descripcion</td>
22                     <td>@item.factor</td>
23                 </tr>
24             }
25     </tbody>
26 </table>
```

Figura 2.15. Código fuente de la vista `Index`

En las líneas de código del 18 al 24, se itera sobre los elementos de la lista que se especificó como modelo para la vista. Para cada elemento en la lista, se genera una fila de tabla HTML con tres columnas.

Las columnas corresponden a las propiedades `idTipoLego`, `descripcion` y `factor`.

Figura 2.16. Interfaz de usuario de la vista `Index` asociada al controlador `tblImagens`

Para el *stub* del subsistema de almacenamiento, se implementa a partir del controlador `tblImagens`, la vista `Index` de la Figura 2.16. Esta interfaz se encarga de presentar una lista con los valores de la tabla `tblImagen`. Permite ingresar una imagen y almacenarla en la base de datos.

```
1
2  @foreach (var item in Model)
3  {
4      <tr>
5          <td> @Html.DisplayFor(modelItem => item.alto)</td>
6          <td>@Html.DisplayFor(modelItem => item.ancho)</td>
7          <td>
8              @if (item.verificador == 1)
9              {
10                 <p>Es lego</p>
11             }
12             else if(item.verificador ==2)
13             {
14                 <p>CLASIFICAR</p>
15             }
16             else
17             {
18                 <p>No Es Lego</p>
19             }
20         </td>
21         <td>
22             <img width="100" src="@Url.Action("show","tblImagens",new { id =item.idImagen})"
   class="rounded mx-auto d-block img-fluid" alt="Responsive image">
23         </td>
24     </tr>
25 }
```

Figura 2.17. Fragmento de código de la vista `Index`

En la Figura 2.17 presenta un fragmento de código de la vista `Index`, asociada al método `Index` del controlador `tblImagens`. En las líneas de código del 7 al 20, para el atributo

verificador se evalúa una condición en donde, si el valor es 2, se indicará como estado `CLASIFICAR` que servirá para el *stub* del subsistema de clasificación y si el valor es 0, se indicará como estado que `No es Lego`.

La línea de código 22, genera una etiqueta de imagen HTML que presentará una imagen en la página web. La imagen se carga utilizando el método `show` del controlador `tblImagens` y se especifica mediante su identificador `id` en la URL de la imagen.

El método de acción `show` en el controlador `tblImagens`, es responsable de recuperar la imagen de la base de datos y devolverla como un archivo de imagen en la respuesta HTTP. La imagen tendrá un ancho de 100 pixeles.

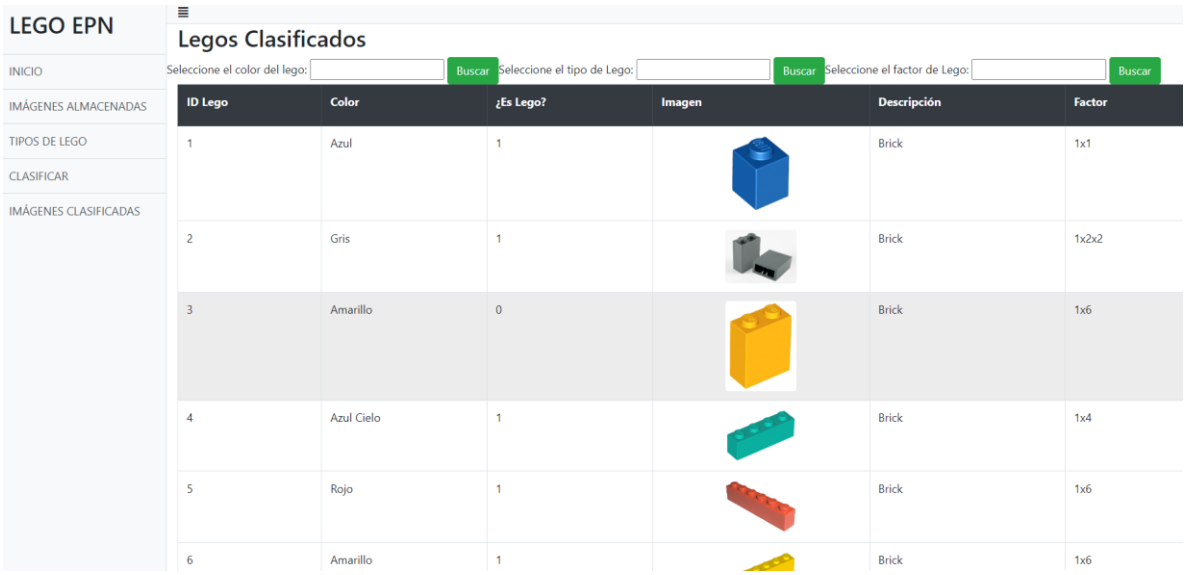

Figura 2.18. Interfaz de usuario de la vista `Index` asociada al controlador `Lego`

La Figura 2.18 presenta la interfaz de usuario generada por la vista `Index` correspondiente al método `Index` del controlador `Lego`. Esta vista se encarga de presentar el listado de la información de la Imagen con la Información producida por el subsistema de clasificación y la información almacenada por el subsistema de almacenamiento.

El código de la vista `Index` se presenta en la Figura 2.19. Se puede apreciar que es similar a la vista de `Index` del método de acción `Index` correspondiente al controlador `TipoLego`.

Adicionalmente se incluye el campo de `imagen,` el cual estará presente en el listado y su implementación es igual a la línea 22 de la Figura 2.17 correspondiente a la vista `Index` del método de acción `Index` del controlador `tblImagens`.

```
<table class="table table-bordered table-hover">
    <thead class="thead-dark">
        <tr>
            <th>@Html.LabelFor(p => Model[0].idLego)</th>
            <th>@Html.LabelFor(p => Model[0].color)</th>
            <th>@Html.LabelFor(p => Model[0].verificador)</th>
            <th>@Html.LabelFor(p => Model[0].imagen)</th>
            <th>@Html.LabelFor(p => Model[0].descripcion)</th>
            <th>@Html.LabelFor(p => Model[0].factor)</th>

        </tr>
    </thead>
    <tbody>
        @foreach(var item in Model)
        {
            <tr>
                <td>@item.idLego</td>
                <td>@item.color</td>
                <td>@item.verificador</td>
                <td>
                    @*<td>@item.idImagen</td>*@
                    <img width="100" src="@Url.Action("show","Lego",new { id
=@item.idImagen})" class="rounded mx-auto d-block img-fluid" alt="Responsive image">
                    @*<td>@item.imagen</td>*@
                </td>
                <td>@item.descripcion</td>
                <td>@item.factor</td>
            </tr>
        }
    </tbody>
</table>
```

Figura 2.19. Fragmento de código de la Vista `Index`

Para el *stub* correspondiente al subsistema de clasificación, a partir del método `Clasificar` perteneciente al controlador `Clasificar` se genera la vista `Index` que se aprecia en la Figura 2.20.

Figura 2.20. Interfaz de usuario del *stub* de clasificación de imágenes

En la interfaz se presentará una lista de imágenes almacenadas por el subsistema de adquisición. Si valor del atributo `verificador` de la tabla `tblImagen` es igual a 2,

corresponde al Estado `CLASIFICAR` el cual hará que se muestre en la interfaz de usuario `CLASIFICAR`. Por medio del botón clasificar, el *stub* asignará valores aleatorios para la tabla `tblTipoLego` y el verificador de la tabla `tblImagen`.

```
1 <td>
2     @Html.ActionLink("Clasificar", "Clasificar", new { id = item.idImagen },
3                      new {  @class="glyphicon glyphicon-edit btn btn-primary"})
4     @Html.ActionLink("Detalles", "Details", new { id = item.idImagen })
5 </td>
```

Figura 2.21. Fragmento de la vista `Index` del controlador `Clasificar`

La Figura 2.21 presenta un fragmento del código de la vista Index correspondiente al método de acción Clasificar del controlador Clasificar. En la línea 2, se utiliza el método `ActionLink` de la clase `HtmlHelper` para crear un enlace que se encargar de dirigir al método de acción `Clasificar` del controlador `Clasificar`, con un parámetro `id` establecido cuando se haga clic sobre el botón.

3 PRUEBAS, RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presenta los resultados obtenidos en las pruebas de funcionamiento. También se presenta una serie de conclusiones y recomendaciones conseguidos en este Trabajo de Integración Curricular.

3.1 Pruebas

Por medio de los *stubs* del subsistema de Clasificación y Almacenamiento se ha procedido a verificar el funcionamiento del subsistema de consulta. Para ello se ha ingresado información en la base de datos la cual permite verificar el funcionamiento del *stub* del subsistema de almacenamiento y por medio de consultas verificar el funcionamiento del subsistema de consulta. Adicionalmente existe la opción de agregar y eliminar imágenes para realizar pruebas con la base de datos.

También se realiza consultas por medio de atributos y así obtener información filtrada que prueba la funcionalidad del subsistema de consulta y el subsistema de clasificación.

3.2 Resultados

Los resultados de las pruebas realizadas en el subsistema de consulta y el los *stubs* del subsistema de almacenamiento y clasificación se presentan a continuación.

Figura 3.1. Interfaz de inicio de la aplicación

En la Figura 3.1 se puede verificar la interfaz de inicio de la aplicación tal y como se estableció en el *sketch* planteado en la metodología. De lado izquierdo se presenta un menú que presentará las demás vistas y este diseño será compartido con las demás interfaces.

En la Figura 3.2 se presenta la información contenida por la tabla tblImagen en la base de datos.

Results | Messages

	idImagen	alto	ancho	verificador	imagen
1	3	173	160	1	0x89504E470D0A1A0A0000000D4948445200000A0000000A...
2	4	449	599	1	0x89504E470D0A1A0A0000000D4948445200000257000001C...
3	5	202	160	0	0x89504E470D0A1A0A0000000D4948445200000A0000000C...
4	6	150	202	1	0x89504E470D0A1A0A0000000D4948445200000CA0000009...
5	7	150	200	1	0x89504E470D0A1A0A0000000D4948445200000C80000009...
6	8	132	202	1	0x89504E470D0A1A0A0000000D4948445200000CA0000008...
7	9	188	202	0	0x89504E470D0A1A0A0000000D4948445200000CA000000B...
8	10	276	250	1	0x89504E470D0A1A0A0000000D4948445200000FA0000011...
9	11	151	202	0	0x89504E470D0A1A0A0000000D4948445200000CA0000009...
10	12	151	202	0	0x89504E470D0A1A0A0000000D4948445200000CA0000009...
11	13	450	599	1	0x89504E470D0A1A0A0000000D4948445200000257000001C...
12	14	398	502	1	0x89504E470D0A1A0A0000000D4948445200000001F60000018...
13	15	144	202	1	0x89504E470D0A1A0A0000000D4948445200000CA0000009...
14	16	120	202	1	0x89504E470D0A1A0A0000000D4948445200000CA0000007...
15	17	189	202	1	0x89504E470D0A1A0A0000000D4948445200000CA000000B...
16	18	3436	4116	1	0xFFD8FFE000104A4649460001010101012C012C0000FFE10E...

Figura 3.2. Campos de la tabla `tblImagen` desde la base de datos

La Figura 3.3 se verifica un listado obtenido desde la base de datos, conformado por una imagen y sus atributos correspondientes verificando así la conexión con la base de datos y contrastando la información que se presenta en la Figura 3.2. Esta Interfaz de usuario corresponde a la viste `Index` del controlador `tblImagens`.

Figura 3.3. Interfaz de usuario generada por la vista `Index` del controlador `tblImagens`

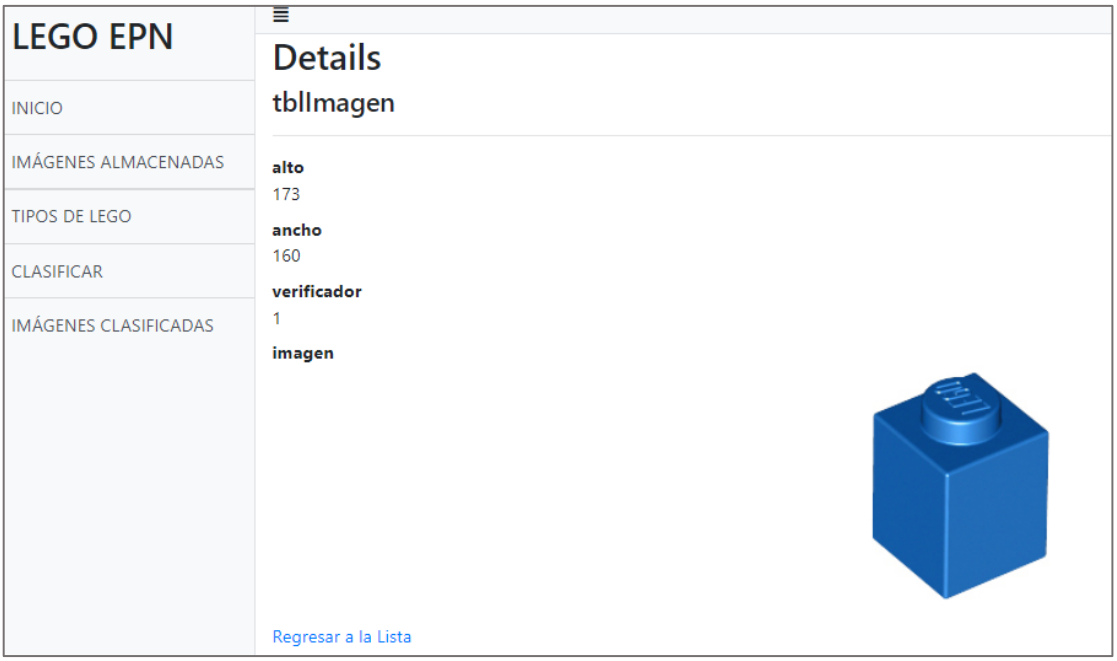

Figura 3.4. Interfaz de usuario que de los detalles de una imagen seleccionada

La Figura 3.4, presenta como se puede acceder a la información individual de cada una de las imágenes.

Manteniendo este formato, la figura 3.5 presenta la posibilidad de agregar imágenes. Esta opción permite verificar el funcionamiento del *stub* de almacenamiento y del *stub* de clasificación.

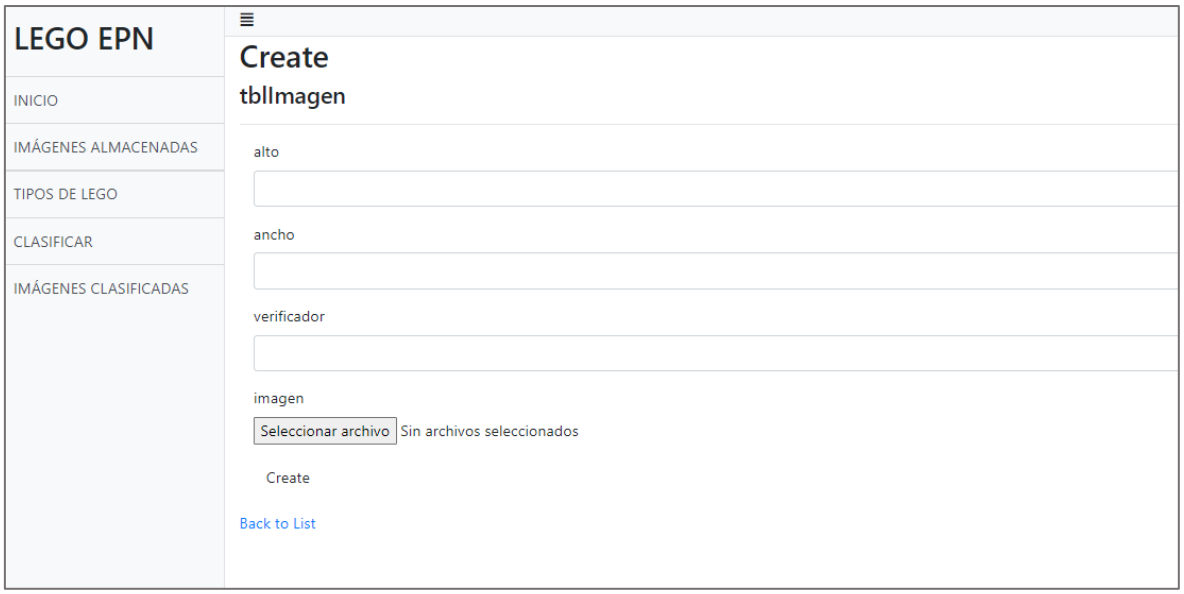

Figura 3.5. Interfaz de usuario para ingresar una nueva imagen

La interfaz de usuario de la Figura 3.6, presenta un listado del contenido de la tabla `tblTipoLego` de la base de datos. La información para el campo `Descripción` y `Factor` ha sido obtenida desde el sitio web Valbrick [10].

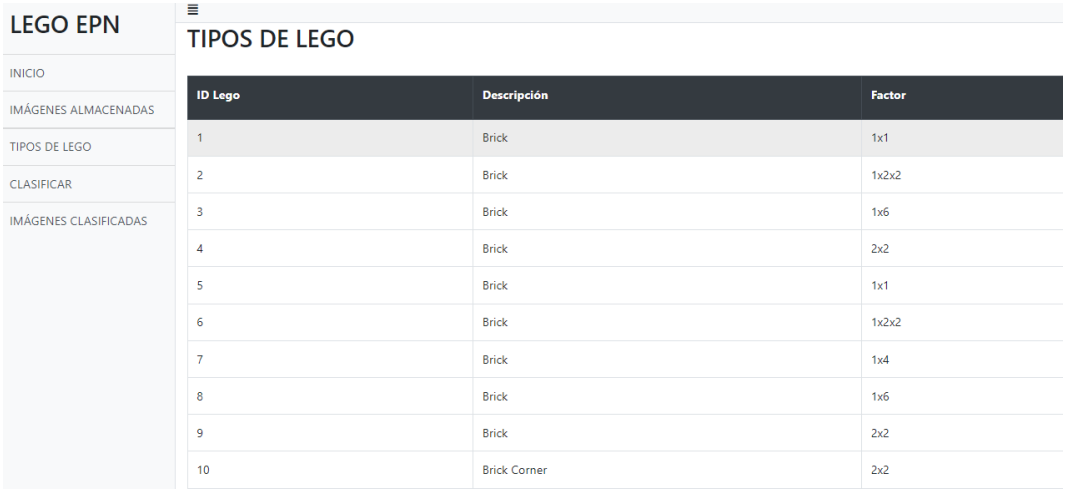

Figura 3.6. Interfaz de usuario para listar los tipos de Lego

La interfaz de usuario presentada en la Figura 3.7, permite realizar la búsqueda de información por parámetros acorde al color, tipo y factor de las fichas Lego que han sido clasificadas y almacenadas.

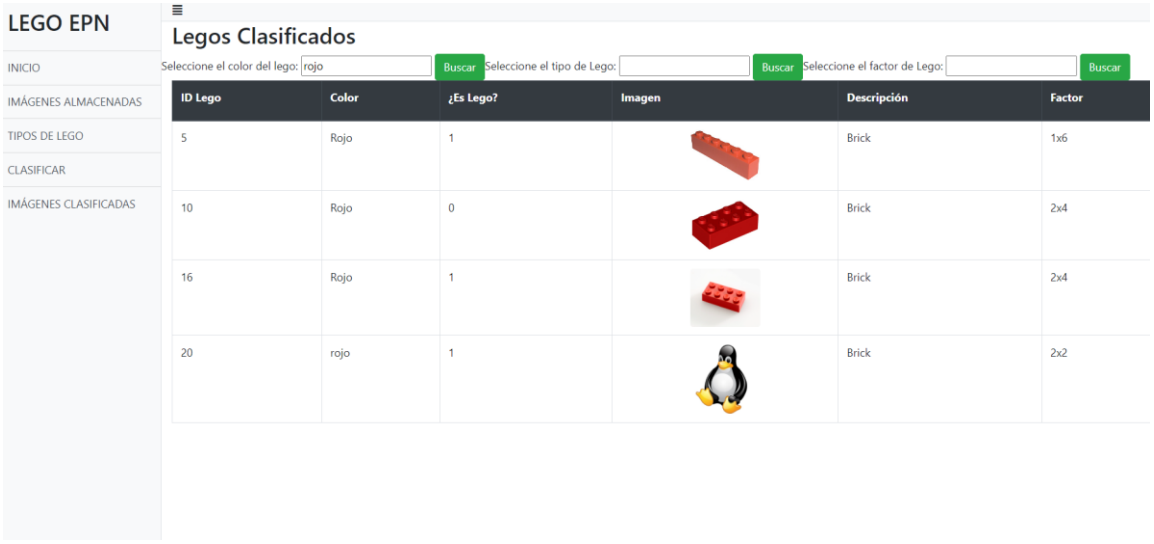

Figura 3.7. Resultado de la búsqueda por color

Para el *stub* del subsistema de clasificación, en la Figura 3.8, se puede verificar como una imagen puede ser considerada o no como ficha Lego.

Figura 3.8. Interfaz de usuario para el *stub* del subsistema de clasificación

En la Figura 3.8, luego hacer clic en el botón azul `Clasificar`, presentará la interfaz de usuario donde se listan las imágenes almacenadas como se puede ver en la Figura 3.9. Se verificar si la imagen fue o no considerada como ficha Lego.

De ser positivo el resultado, también estará presente en la interfaz de usuario de la Figura 3.9 y con sus respectivos atributos como presenta la Figura 3.10. Caso contrario no lo hará.

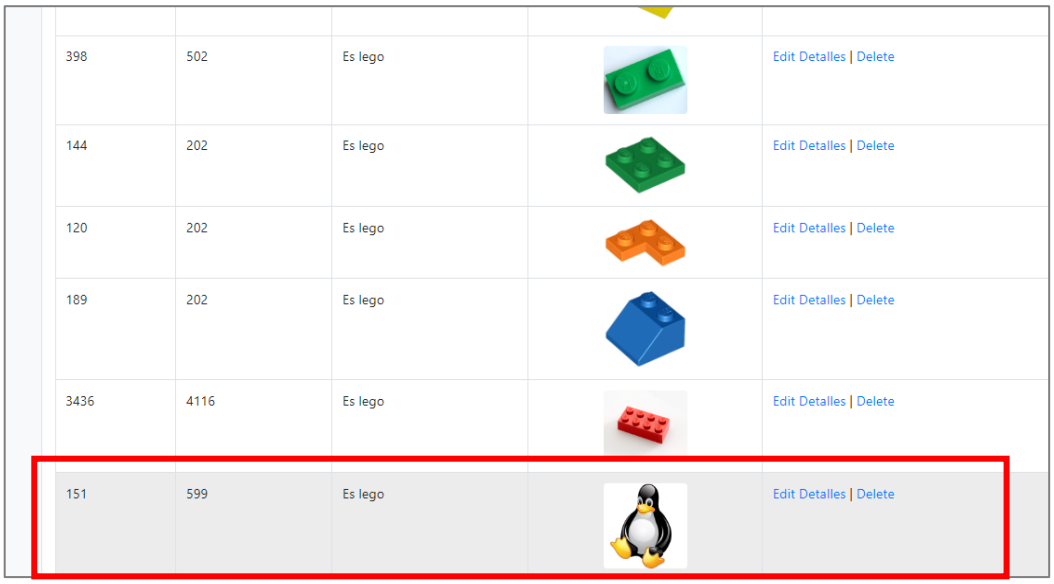

Figura 3.9. Indica que la imagen se considera como ficha Lego

12	Verde	1		Plate	1x2
13	Verde	1		Plate	2x1
14	Naranja	1		Plate Corner	2x2
15	Azul	1		Slope	2x2
16	Rojo	1		Brick	2x4
20	rojo	1		Brick	2x2

Figura 3.10. Se presenta la imagen considerada como ficha Lego listada en las imágenes clasificadas

El comportamiento responsivo de la aplicación se puede verificar en el navegador web en donde se está ejecutando la aplicación.

Gracias al *framework* Bootstrap implementado en el subsistema de clasificación se logra obtener el comportamiento adecuado. Se hace la prueba con las dimensiones de un iPad Air como presenta la Figura 3.11.

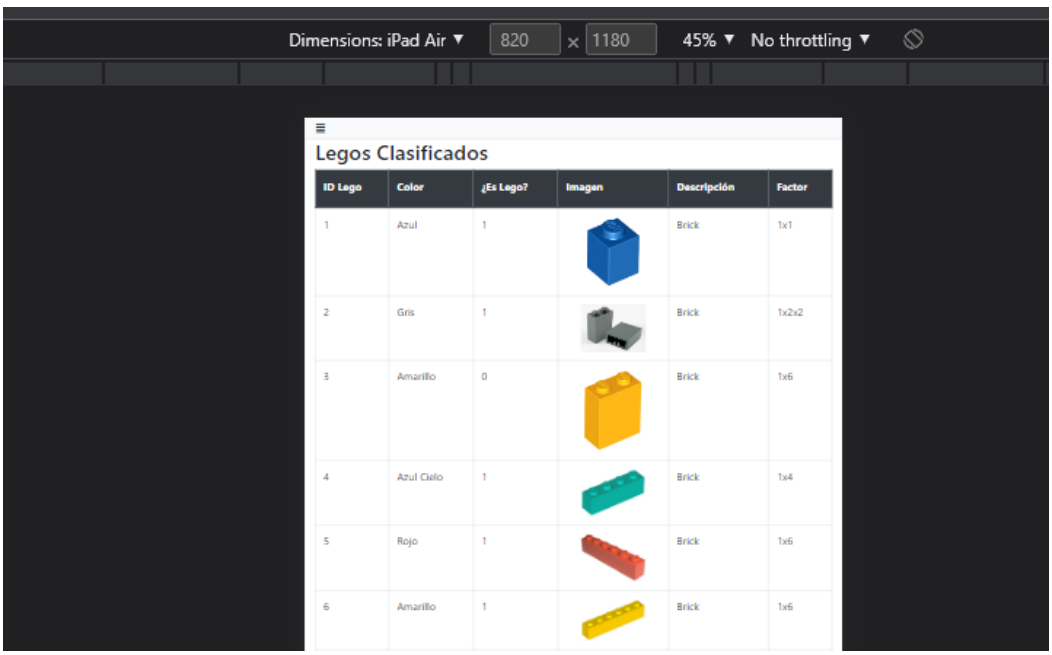

Figura 3.11. Comportamiento responsivo de la aplicación

3.3 Conclusiones

- Al finalizar el presente Trabajo de integración Curricular se dispone de un subsistema de consulta, que es parte del sistema de clasificación de fichas Lego basada en imágenes.
- Para el desarrollo del subsistema de consulta del presente Trabajo de Integración Curricular se usa la arquitectura cliente – servidor y el *framework* ASP.NET MVC.
- Se dispone de una base de datos relacional para simular el subsistema de almacenamiento, el cual contiene la información de las imágenes y almacenará información de las fichas Lego.
- Se utiliza Entity Framework con el enfoque Database First para gestionar la información almacenada en la base de datos.
- Al utilizar Entity Framework se aprovecha la característica de integración con LINQ, para el intercambio de información por medio consultas que simplifican y mejoran el proceso de desarrollo del subsistema.
- Se verificó que Razor permite utilizar el lenguaje de programación C# en conjunto con HTML de manera sencilla, obteniendo vistas dinámicas y personalizadas que se adaptan a las necesidades del usuario.
- Al establecer un diseño de vista compartida se evitó el codificar de manera repetida código y así optimizar el tiempo. Además, el código implementado en la vista es más entendible al momento de identificar algún error de codificación.
- El implementar el *framework* Bootstrap en el subsistema permitió que la aplicación se adapte a cualquier tipo de pantalla. Esto se puede verificar en el navegador configurando el tamaño de la pantalla a diseños de varios dispositivos y poder simularlos.

3.4 Recomendaciones

- El acceso a las imágenes almacenadas en la tabla tblImagen mejoraría si, en lugar de almacenar la imagen en la base de datos, se almacenaría la ruta de la imagen que estaría almacenada en un computador o servidor y así evitar el convertir la imagen a otro formato.

- Se recomienda no modificar los modelos creados por Entity Framework para evitar conflictos dentro del desarrollo del subsistema de consulta. Es recomendable trabajar con modelos adicionales manteniendo la cadena de conexión.
- Para la base de datos, se recomienda trabajar con *seeders* para disminuir el ingreso manual de información. Se reduciría el porcentaje de error en la información digitada por el ser humano.
- Se recomienda desplegar la aplicación en una red LAN o WAN para realizar pruebas siempre y cuando se cuente con la infraestructura necesaria para llevar a cabo esta actividad.
- Para un mejor renderizado de las interfaces se recomienda trabajar con la última versión del *framework* Bootstrap ya que presenta mejoras que van de la mano con las actualizaciones de los navegadores web.
- Adicionalmente se recomienda trabajar con el *Grid* de Bootstrap para que el diseño sea más ordenado y la adaptabilidad a los distintos tamaños de pantalla sea óptimo.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] E. C. Navone, «freeCodeCamp,» 09 11 2022. [En línea]. Disponible en: https://www.freecodecamp.org/espanol/news/aprende-bootstrap-5-en-espanol-creando-tu-portafolio-personal-curso-de-bootstrap-desde-cero/. [Último acceso: 10 12 2022].

[2] R. anderson, «Microsoft,» Microsoft, 21 09 2022. [En línea]. Disponible en: https://learn.microsoft.com/es-es/aspnet/mvc/overview/getting-started/introduction/adding-a-controller. [Último acceso: 11 12 2022].

[3] M. M. Serafin, Programador Jr. de Aplicaciones ASP.NET MVC, 2018.

[4] R. R. Singh, Mastering Entity Framework, Birmingham: Packt Publishing Ltd, 2015.

[5] A. Vickers, «Microsoft Learn,» 28 09 2022. [En línea]. Disponible en: https://learn.microsoft.com/es-es/ef/ef6/modeling/designer/workflows/database-first. [Último acceso: 28 02 2023].

[6] P. A. M. 4, Jess Chadwick, Todd Snyder, Hrusikesh Panda, United States of America: O´Reilly, 2012.

[7] F. Karaben, «Getbootstrap,» 13 08 2022. [En línea]. Disponible en: https://getbootstrap.esdocu.com/docs/5.1/extend/icons/. [Último acceso: 25 01 2023].

[8] H. Kniberg, Kanban y Scrum - obteniendo lo mejor de ambos. Prologo de MAry Poppendieck & David Anderson, Estados Unidos de América: C4Media Inc, 2010.

[9] J. S. Ken Schwaber, La Guía de Scrum, 2013.

[10] Valbrick, «Vocabulario Lego capítulo 2.,» valbrick, 20 11 2018. [En línea]. Disponible en: https://valbrick.com/2018/11/20/vocabulario-lego-capitulo-2-brick-stud-plate/. [Último acceso: 23 02 2023].

5 ANEXOS

ANEXO I: *Script* de la base de datos.

ANEXO II: *Sketches* de las interfaces de usuario.

ANEXO III: Solución del subsistema de consulta (Código Fuente).

ANEXO IV: Resultados de la encuesta sobre el uso del subsistema de consulta.

En el CD adjunto al documento, se incluye todos los anexos debido a su extensión.